CA2248634

Publication Title:

COMMON CONNECTOR FRAMEWORK

Abstract:

Abstract of CA2248634

An object oriented framework for communication between an application component running in an infrastructure and a backend system. The framework includes a connector protocol definition for a implementation by a connector component. The connector component enables and constrains communication between the application component and the backend system. The framework provides client view interface definitions for communication between the application component and the connector component, and infrastructure view interface definitions for communication between the infrastructure and the connector component. Data supplied from the esp@cenet database - Worldwide

---

Courtesy of http://v3.espacenet.com

This Patent PDF Generated by Patent Fetcher(TM), a service of Stroke of Color, Inc.

(19) Canadian Intellectual Property Office
An Agency of Industry Canada

Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

(11) CA 2 248 634 (13) A1
(43) 24.03.2000

(12)

(21) 2 248 634

(22) 24.09.1998

(51) Int. Cl.[6]: H04L 029/02, H04L 012/46

(71) IBM CANADA LIMITED-IBM CANADA LIMITEE, 3600 Steeles Avenue East, MARKHAM, O1 (CA).

(72) PRZYBYLSKI, Piotr (CA).
BEISIEGEL, Michael (CA).
STARKEY, Michael (CA).

(74) HOICKA, LEONORA

(54) CHASSIS COMMUN DE CONNEXION
(54) COMMON CONNECTOR FRAMEWORK

(57)

An object oriented framework for communication between an application component running in an infrastructure and a backend system. The framework includes a connector protocol definition for a implementation by a connector component. The connector component enables and constrains communication between the application component and the backend system. The framework provides client view interface definitions for communication between the application component and the connector component, and infrastructure view interface definitions for communication between the infrastructure and the connector component.

OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

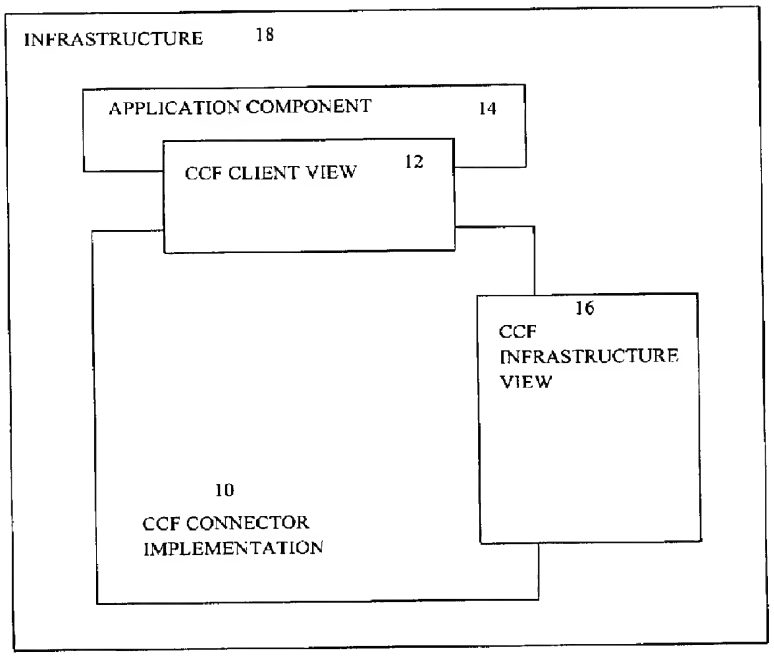
(12) (19) (CA) **Demande-Application**

(21) (A1) **2,248,634**
(22) 1998/09/24
(43) 2000/03/24

(72) PRZYBYLSKI, Piotr, CA
(72) BEISIEGEL, Michael, CA
(72) STARKEY, Michael, CA
(71) IBM CANADA LTD. - IBM CANADA LIMITÉE, CA

(51) Int.Cl.[6] H04L 29/02, H04L 12/46

(54) **CHASSIS COMMUN DE CONNEXION**
(54) **COMMON CONNECTOR FRAMEWORK**



(57) An object oriented framework for communication between an application component running in an infrastructure and a backend system. The framework includes a connector protocol definition for a implementation by a connector component. The connector component enables and constrains communication between the application component and the backend system. The framework provides client view interface definitions for communication between the application component and the connector component, and infrastructure view interface definitions for communication between the infrastructure and the connector component.

COMMON CONNECTOR FRAMEWORK

ABSTRACT

An object oriented framework for communication between an application component running in an infrastructure and a backend system. The framework includes a connector protocol definition for a implementation by a connector component. The connector component enables and constrains communication between the application component and the backend system. The framework provides client view interface definitions for communication between the application component and the connector component, and infrastructure view interface definitions for communication between the infrastructure and the connector component.

CA9-98-028

COMMON CONNECTOR FRAMEWORK

FIELD OF THE INVENTION

The present invention is directed to an improvement in computing systems and in particular to a framework providing common connections between computer systems and computer software.

BACKGROUND OF THE INVENTION

A large number of software products are designed to be used to permit the connection different computer systems running on various hardware platforms and under different software configurations. This type of connecting software is referred to as "middleware", and is used to provide clients (application programs or their components) with an access to different types of systems, such as transaction servers or database systems (referred to as "backend systems").

Middleware is often provided as a library which is specific to a particular programming language and operating system. The middleware library provides to client programs a well defined set of APIs used to access the middleware and the backend system which the client is seeking to communicate with.

There are several problems with this middleware approach. For the middleware implementors, the main problem is the variety of the backend computer systems that middleware has to use. For each such computer system, often very different in its support and services, the middleware has to provide consistent behaviour e.g. error reporting mechanism, often having to implement its services multiple times.

The main problem for the clients is the number of different APIs that even a simple client may be

required to support. Each middleware product has its own requirements, and a client that is designed to use several backend systems will typically be required to communicate using more than one piece of middleware. The result is that the client program or system has to support an ever growing number of specific APIs. The more versatile and powerful the client becomes, the more complexity is associated with the client's access to backend systems and the resulting expanded use of middleware systems. Even more severe problems may potentially occur when the client constitutes a part of a tool or a framework. In that case maintaining flexibility in the tool or framework comes at the cost of increasing the client, and therefore tool or framework, complexity.

It is therefore desirable to have a set of interfaces, or protocols, which will define how client programs access backend systems. Such a set of interfaces, or protocols, will permit clients to access backend systems using the same set of APIs independent of the concrete implementation of those APIs. It is desirable to have such interfaces or protocols implemented as a framework, which is able to utilize object oriented programming and design concepts to provide a formal definition of the set of interfaces so as to permit middleware to be replaced with a tool which will permit a uniform approach to accessing backend systems.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, there is provided an improved computer system for client to backend system communication.

According to another aspect of the present invention, there is provided an object oriented framework for communication between an application component running in an infrastructure and a backend system, comprising a connector protocol definition for a implementation by a connector component for communication between the application component and the backend system, the framework further comprising a client view protocol definition for communication between the application component and the connector component, and an infrastructure view protocol definition for

communication between the infrastructure and the connector component.

According to another aspect of the present invention, there is provided the above object oriented framework in which the client view protocol definition comprises a connection specification protocol definition to define communication of backend system connection properties to the connector component, an interaction specification protocol definition to define communication of properties associated with application component requests of backend system, a communication specification protocol definition to define connection, disconnection and execution of communication functions for communication between the application component and the backend system.

According to another aspect of the present invention, there is provided the above object oriented framework in which the infrastructure view protocol definition comprises a set of managed connection protocol definitions to define the creation and management of connections between the application component and the backend system, wherein the set of managed connection protocol definitions comprises a first subset of managed connection protocol definitions for implementation in the connector component and a second subset of managed connection protocol definitions for implementation in the infrastructure, a set of resource coordination protocol definitions to define coordination of resources relating to connections between the application component and the backend system, wherein the set of resource coordination protocol definitions comprises a first subset of resource coordination protocol definitions for implementation in the connector component and a second subset of resource coordination protocol definitions for implementation in the infrastructure, an identifier protocol definition for implementation by the infrastructure to provide connector component access to a unique identifier, a logon information protocol definition for implementation by the infrastructure to provide connector component access to security items in the infrastructure, a component definition for logon information for definition of security items required for connector component communication between the application component and the backend system, a RAS protocol definition for implementation by the infrastructure to provide connector access to error and trace logging.

According to another aspect of the present invention, there is provided the above object oriented framework wherein the application component runs on a thread in an infrastructure providing infrastructure services, the object oriented framework further comprising means for the infrastructure to provide infrastructure services to the connector component by associating a runtime context of the infrastructure services with the thread of the application component

According to another aspect of the present invention, there is provided the above object oriented framework in which the first subset of managed connection protocol definitions comprises a protocol definition for the creation of a connection, and a protocol definition for providing a connection manager component with control over a managed connection, and in which the second subset of managed connection protocol definitions comprises a protocol definition for reserving and releasing managed connections.

According to another aspect of the present invention, there is provided the above object oriented framework in which the first subset of resource coordination protocol definitions comprises a protocol definition for creating a view of a connection resource for use by a resource coordinator, and in which the second subset of resource coordination protocol definitions comprises a protocol definition for registration of resources by a resource coordinator.

According to another aspect of the present invention, there is provided the above object oriented frameworks in which the framework is implemented in the Java programming languages and in which the protocol definitions are interfaces in the Java programming language.

According to another aspect of the present invention, there is provided the above object oriented framework in which the application component runs on a thread in a Java infrastructure, the object oriented framework further comprising means for the infrastructure to provide infrastructure services to the connector component by associating a Java runtime context with the thread of the application component.

According to another aspect of the present invention, there is provided a computer system

comprising a set of tools for use to create a communication between a first computer system component and a second computer system component, the tools constraining the communication to occur using a predefined protocol, the set of tools comprising a connector tool for a implementation by a connector for communication between the first computer system component and the second computer system component, the connector tool comprising a client view tool for defining communication between the application component and the connector component, and an infrastructure view tool for defining communication between the infrastructure and the connector component.

Advantages of the present invention include the provision of a framework to define the interaction between the client and the backend such that the same framework may be implemented for different infrastructures and different backend systems. The need for clients to utilize different APIs, and for infrastructures to support different middleware systems, is therefore reduced.

BRIEF DESCRIPTION OF THE DRAWINGS

The preferred embodiment of the invention is shown in the drawings, wherein:

Figure 1 is a block diagram showing the architecture of the framework of the preferred embodiment of the invention, including an application component.

Figure 2 is block diagram illustration an example of system connections using the framework of the preferred embodiment.

Figure 3 is an interface diagram for the client view of the framework of Figure 1.

Figure 4 is a block diagram showing the architecture of the framework of the preferred embodiment including a fat application.

Figure 5 is an interface diagram for the infrastructure view of the framework of Figure 1.

Figure 6 is a block diagram showing the architecture of the framework of the preferred embodiment, including the runtime context.

In the drawings, the preferred embodiment of the invention is illustrated by way of example. It is to be expressly understood that the description and drawings are only for the purpose of illustration and as an aid to understanding, and are not intended as a definition of the limits of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to Figure 1, there is illustrated in a block diagram the architecture of the framework of the preferred embodiment.

The term "framework" relates to object oriented programming languages and systems. The preferred embodiment is described below with reference to the object oriented language Java. Terminology relating to the Java language is used in the description of the preferred embodiment. It will be appreciated by those skilled in the art that the use of object oriented languages such as Java facilitates the description of the preferred embodiment. However the protocols of the preferred embodiment can be adopted and adapted to in other programming environments. Similarly, the preferred embodiment can be implemented without using frameworks, although there are advantages to both implementing and describing the preferred embodiment using the notion of frameworks.

"Frameworks" are used in this application to mean a collection of classes or interfaces in an object oriented programming language which is designed to guide and constrain how a communication or function is to be carried out by a system making use of the framework. A framework is similar in

certain respects to libraries. However, a library is typically a collection of routines or data definitions which must be subject to relatively extensive programming and data design steps before the subroutines and/or data definitions of the library are usable. In contrast, a framework not only defines data structures and methods but also provides a defined interrelationship which ensures that the methods and data structures are usable with limited input from the user.

The user of a framework is able to use certain of the objects defined in the framework and is able, where permitted by the framework design, to extend certain of the objects defined in the framework. Thus a framework both provides predefined functionality and data and permits the extension of such predefined functions and data by the user of the framework. A framework therefore enables and constrains the user with respect to accessing resources and communicating in computer systems.

By using frameworks, which are capable of tightly defining data, functionality, and interfaces, it is possible to utilize automated code generation facilities to write code to make use of the framework. In contrast, typical middleware solutions to communication between client and backend systems present substantial difficulties in automating the generation of code for use with the APIs of the potentially dissimilar middleware systems.

The framework of the preferred embodiment is described in the Java language, but is capable of being implemented in other object oriented languages.

Figure 1 presents a Common Connector Framework (CCF), in block 10. Associated with the CCF are two sets of interfaces. One is named the CCF Client View (shown as block 12) and is between the client or application component (shown as block 14) and the connector of block 10. The second interface is the CCF Infrastructure View (shown as block 16) which is between the connector of block 12 and the environment specific infrastructure (shown as block 18). The environment that the CCF is running in as shown in Figure 1 is a well-defined component-based infrastructure.

The term "interface" as used in this description refers to Java interfaces. Java interfaces are analogous to other object oriented language constructs referred to as protocols. An interface may

be thought of as a class which defines an agreed-upon behaviour. A Java interface includes definitions of methods (and thus defines what behaviour of a class that implements the interface) but the definitions are null definitions. It is the class that implements the interface which must implement all the methods defined in the interface, thereby "agreeing" to certain behaviour. A class is able to implement more than one interface (for this reason, it is possible to view interfaces as supporting multiple inheritance).

Returning to Figure 1, the division of the CCF into the two separate interfaces of blocks 12 and 16 allows for flexibility of implementation and use. The architecture of the framework of the preferred embodiment is designed to permit various clients (or application components) to use well-defined and consistent interfaces to create the connection with the desired backend systems. By dividing the CCF connector interfaces into a client view interface and an infrastructure interface, the client can be isolated from the details of how a particular infrastructure handles such functions as error handling, and trace logging (RAS services), or security controls.

The framework of the preferred embodiment permits all clients using the framework to utilize the same interface definition to obtain access to all backend systems. The CCF connector will be different for each specific backend system but the defined client and infrastructure interfaces permit each CCF connector to provide a well-defined structure for communication between the client and the backend system. As is set out below, the interfaces defined in the preferred embodiment are sufficiently simple to provide ease of use, yet are sufficiently powerful to be useable by all manner of client and backend system combinations.

The set of interfaces shown in Figure 1 constitutes a contract between the client block 14, the connector block 10 and the platform specific infrastructure block 18. The set of interfaces is small enough to allow straightforward implementation, yet flexible enough to accommodate different requirements of components. As will be seen from the following description, a client that uses different connectors does not need to interface with numerous APIs of different middleware systems. Instead, the client is able to use the same interface calls to the connector to establish a connection,

execute requests and disconnect from the backend system, with the differences between connectors being limited to two well defined interfaces.

The architecture shown in Figure 1 also permits the connector implementation to be independent of the platform it runs on (the infrastructure 18 in Figure 1), since it can request services, such as error reporting, security information or coordination, from the infrastructure by using well defined set of interfaces. Similarly, infrastructure 18 does not need to be concerned with the details of any particular connector implementation and can implement generic services such as connection management and coordination that can be used by any CCF connector.

A CCF connector is a package of classes which include implementations of the interfaces in the CCF (those interfaces indicated as being implemented by the connector in Figures 3 and 5, below). The classes in the package of a particular CCF connector are written to permit communication with a particular backend system.

Figure 2 is a block diagram which illustrates how different clients are able to use implementations of CCF connectors to establish communication with backend systems. In the example of Figure 2, there are two different infrastructures, shown as infrastructure n in block 20 and infrastructure p in block 22. Clients are shown as Client1, Client2, Client3, and Client4 in blocks 24, 26, 28, and 30, respectively. Client1 and Client2 run in infrastructure n, while Client3 and Client4 run in infrastructure p. The two backend systems are illustrated as BackendA and BackendB, in blocks 32 and 34, respectively.

The CCF connector implementations to permit the clients of blocks 24 to 30 to communicate with the backend systems of blocks 32 and 34 are shown as blocks 40, 42, 44 and 46. Block 40 represents a CCF connector implementation which is written for BackendA. This package contains classes which include implementations of interfaces ConnectionSpec, InteractionSpec, Communication, Managed Factory, Managed, Resource, and an extension of the class LogonInfoItems. These interfaces and this class are described below.

Connector implementation 40 has available to it the infrastructure services of infrastructure n. This is accomplished by use of the Infrastructure View interface, as implemented for infrastructure n. The Infrastructure View interface is described in more detail below.

Similarly, Client3 uses CCF connector implementation 44 for BackendA, which also makes use of the Infrastructure View interface to obtain the infrastructure services of infrastructure p (block 22).

In a like manner, Client2 and Client4 make use of CCF connector implementations designed for BackendB, which connector implementations run in infrastructures n and p, respectively. Client2 also makes use of the CCF connector implementation for BackendA. In this case Client2 must create a separate communication with Connector A_n.

As can be appreciated, the clients of the example of Figure 2 will make use of the common interface provided in the CCF Client View interface to carry out the communication with the two backend systems. The CCF connector implementations for the two backend systems will likewise make use of the common interface provided in the CCF Infrastructure View interface to obtain access to the necessary services of the infrastructures.

As referred to above, the framework of the preferred embodiment includes interfaces which themselves define methods. The following describes the interfaces in the framework of the preferred embodiment, including which components implement and which components use the interfaces, and which methods are defined in the interfaces.

Figure 3 is an interface view which describes the interfaces of the Client View of the Common Connector Framework. The Client View represents the contract between the thin client or application component and the connector implementation. The following interfaces define the connector to the client:

1. Communication (block 50 in Figure 3) is an interface implemented by all CCF connectors. This interface defines methods to establish connection with the target system, perform the

communication and terminate the communication. The interface defines the following methods (set out in block 52):

- public abstract void connect()
  This method establishes the connection with the backend system and if necessary can also perform any communication specific setup such as verification of the validity of the client's security credentials.

- public abstract void execute (InteractionSpec interactionspec, Object input, Object output)
  This method performs one interaction with the backend system. The input and output objects depend on the backend resource used.

- public abstract void disconnect ()
  This method terminates the communication with the back end system and performs the necessary cleanup preparing the communication for the next use.

2. ConnectionSpec (block 54) is an interface encapsulating all the connection relevant properties such as host name, port number or the queue name. The ConnectionSpec is able to create the communications and provide a unique identifier, such as hash code, for each connection. This interface defines two methods (in block 56):

- public abstract Communication createcommunication()
  This method creates a new communicaton.

- public abstract int hashcode()
  This method calculates and returns a connection specific unique identifier based on the connection properties (for example, host name, queue name).

3. InteractionSpec (block 58) is an interface which defines request specific properties such as program or transaction name, the interaction mode defining whether the request is synchronous or asynchronous (SEND-RECEIVE, SEND, RECEIVE), and other connector specific properties.

As Figure 3 indicates, the CCF connector implements the interfaces of the CCF Client View, and the client uses the interfaces. The methods set out in the interfaces of the CCF Client View are sufficiently generalized and powerful to permit the Client to access the backend system which a particular connector is defined to connect to. As will also be appreciated, the client using the connector of the CCF does not have to deal with those infrastructure functions such as RAS services, which may be dealt with by the CCF connector in the Infrastructure View. Thus error handling functions, for example, which are expressly dealt with by clients using the typical middleware product, are no longer necessarily handled at the client level.

However, in some cases the client is designed so that the client itself uses the infrastructure services which are set out in the Infrastructure View interface. Figure 4 presents a view of the CCF by such a fat application client. In Figure 4 CCF Connector implementation 70 is shown having CCF Client View 72 and CCF Infrastructure View 74. There is no infrastructure shown as a separate block in the block diagram of Figure 4. Instead, Fat Application 76 is shown. This type of more complex client uses the infrastructure interface of CCF Infrastructure View 74 for a direct access to the services such as coordination or security. As will be appreciated, the architecture of Figure 4 is a special case of the Figure 1 architecture of the preferred embodiment. The description below will focus on the Figure 1 architecture, but the preferred embodiment may be implemented with necessary modifications to provide the architecture of Figure 4.

An example of how the interfaces defined above may by implemented in classes which are used by a client may be seen in reference to the example set out in Figure 2. Client1 in Figure 2 makes use of ConnectorA_n. ConnectorA_n is a package which contains classes which implement interfaces as defined above. In this example, assume that ConnectorA_n implements the ConnectionSpec

interface by class A_nConnectionSpec, InteractionSpec by class A_nInteractionSpec, and Communication by class A_nCommunication.

The client may then create a new object using the following code:

```
Client1_ConnectionSpec = new ConnectorA_n.A_nConnectionSpec
```

The object Client1_ConnectionSpec is an instance of the ConnectionSpec interface implementing class A_nConnectionSpec, as found in the package ConnectorA_n.

A communication object may then be created by the client by using the CreateCommunication method which is found in the A_nConnectionSpec class (mandated by the ConnectionSpec interface):

```
A_nCommunication c = Client1_ConnectionSpec.CreateCommunication ()
```

An interaction specification must also be created by the client. The following code creates the object Client1_InteractionSpec:

```
A_nInteractionSpec Client1_InteractionSpec =
                    new ConnectorA_n.A_nInteractionSpec
```

The client may then define the properties of the Client1_InteractionSpec object to reflect what interaction is desired with the backend system. The description of the ConnectorA_n package will also indicate what object definition is required for input and output to and from the backend system. The client uses `c.connect()` to create a connection. The client may then use the execute method, as defined in the class A_nCommunication which implements the Communication interface in package ConnectorA_n. For example:

```
CALL c.execute (A_nInteractionSpec Client1_InteractionSpec,
                InputObject input, OutputObject output)
```

The above example illustrates, the manner in which the interfaces of the CCF Client View are used to permit a client to communicate with a backend system. In the example, the call to the c.execute

method ensures that the interaction desired by the client will be carried out on the input in a manner as defined in the interaction specification and that the output will receive the result of that interaction.

In Figure 5, the Common Connector Framework Infrastructure View is illustrated. As described above, the CCF Infrastructure View represents the contract between the implementation of a CCF connector and a platform specific infrastructure. The interface between the CCF connector and the infrastructure differs from the contract between the thin client and the connector. In the latter, the connector implements and the client uses the set of interfaces (defined in the Client View interface shown in Figure 3). In the case of the relationship between the CCF connector and the infrastructure, both the connector and the infrastructure have to implement the set of interfaces to be used by the other.

The interfaces set out in the CCF Infrastructure View are shown in Figure 5. The interfaces (and the class) which are implemented by the CCF connector and used by the CCF infrastructure are the ManagedFactory interface 80, Managed interface 82, Resource interface 84, and LogonInfoItems class 86. The interfaces which are implemented by the infrastructure and used by the CCF connector are ConnectionManager 88, Coordinator interface 90, Identifier interface 92, LogonInfor interface 94, and RASService interface 96. The implementations of the interfaces of CCF Infrastructure View are designed to interact with the implementations of interfaces which are found in the CCF Client View (as shown in Figure 3).

The interfaces (and class) of the CCF Infrastructure View which are implemented by a CCF connector are designed as follows:

1. ManagedFactory interface 80 provides a connection manager (an implementation of ConnectionManager interface 82) with the generic way of creating connections from ConnectionSpec 54. The ManagedFactory interface 80 defines the following method:

- public Managed createManaged (ConnectionSpec connectionspec)

This method creates a managed connection based on the connection specification (from the CCF Client View ConnectionSpec 54).

2. Managed interface 82 is implemented by a managed connection. A connection manager (implementation of ConnectionManager interface 88) is able to control the entire life cycle of the managed connection from its creation by an implementation of ManagedFactory interface 80 to its destruction using the destroyManaged method which is defined in Managed interface 82. Managed connections can also be reused by the connection manager and therefore methods are included which relate to the reuse of a managed connection. The main methods of Managed interface 82 are:

- public void destroyManaged ()

This method allows a connection manager to destroy a managed connection. The method is used, for example, in cases where the connection manager reduces the number of the connections managed by it.

- public void prepareManagedForReuse ()

This method prepares the connection to be reused by the connection manager.

3. Resource interface 84 defines the coordinator view of the connection resource. It contains resource definition similar to that found in JTS/OTS and includes one and two phase commit protocol support. Resource interface 84 defines the following methods:

- public void commit ()

The coordinator tells the resource that its state is valid and it is no longer needed.

- public void rollback()

The coordinator tells the resource that its state is invalid and it is no longer needed.

- public void commitOnePhase ()

The coordinator tells the resource that it can commit work using a one phase commit

protocol.

- public int prepare ()
The coordinator asks the resource whether its state is valid and if a subsequent call to commit would succeed.

In addition to the interfaces defined as described above, the connector extends and uses the following class which is in the CCF Infrastructure View:

4. LogonInfoItems class 86 is the superclass for all connector specific LogonInfoItem classes. It defines the way in which a given CCF connector accesses security items specific to that connector.

- public logonInfo getlogoninfo ()
This method returns the logonInfo object used to carry out the logoninfoitems access methods. The methods are used in this object.

- public string getpassword ()
This method returns the password.

- public string getuser ()
This method returns the user.

- public void setpassword (string password)
This method sets the password.

- public void setuser (string user)
This method sets the user.

As the above description indicates, the CCF Infrastructure view contains interfaces (and one

superclass) which require that a protocol for managed connections be created when the framework of the CCF is instantiated. When a client wishes to access a backend system, the client will call instances of objects created in classes which implement the interfaces of the CCF Client View. The CCF connector implements the interfaces in the CCF Infrastructure View to permit a managed connection to be created with the backend system. The managed connection makes use of infrastructure services which are made available to the CCF connector object in accordance with the interfaces in the CCF Infrastructure View which are implemented by each particular infrastructure which supports the Common Connector Framework.

Referring again to Figure 5, the interfaces of the CCF Infrastructure View which are implemented by the infrastructure are designed as follows:

1. ConnectionManager interface 88 defines the common connection manager as seen by a CCF connector. ConnectionManager interface 88 defines management of connections based on the hashcode method of ConnectionSpec interface 54. If the infrastructure provides session ID then the connection manager implementing the ConnectionManager interface 88 will manage the connection based on SessionID (see below description of the Identifier interface), as well. The connection manager uses the connection management properties, defined on the ConnectionSpec to create, reuse and destroy connections. ConnectionManager 88 relates to connection management functions which can include timeout requirements, billing constraints, or maximum simultaneous connections, for example.

ConnectionManager interface 88 defines the following methods:

- public Managed reserve (ManagedFactory managedfactory, ConnectionSpec connectionspec)

This method reserves a connection for the caller and returns it for the exclusive use of the caller. Typically this will be called in the connect method (shown in block 52 of Figure 3) of an implementation of Communication interface 50.

- public void clearForSessionID (Identifier aSessionID)

This method clears the used connections list of the coordinator identified by the session.

-public void release (Managed managed)

This method is called to release a connection used exclusively by a client. Typically this will be called by the disconnect method in the implementation of Communication interface 50.

2. Coordinator 90 defines the common coordinator interface as seen by an implementation of the Resource interface of a connector. Coordinator 90 allows registration of resources. Since some resources should be handled before or after the connection resources, the appropriate registration method should be used for such resources (registerBefore or registerAfter). An implementation of the Coordinator interface also returns a unique identifier, defined as a CoordinationID object, that may be used by a backend system which is defined to require such an ID. Coordinator interface 90 is related to coordination of logical units of work and provides a functionality to permit certain resources to be logically grouped such that interactions are appropriately handled to preserve the logical interrelationship of the resources.

Coordinator interface 90 defines the following methods:

- public void clearAllRegistered ()

This method disposes of all the resources registered with the coordinator.

- public Identifier getCoordinationID ()

This method returns the unique Coordination Identifier of this coordinator. The

CoordinationID is not changed by commit or rollback methods.

- public void register(Resource resource)
This method registers the Resource with the coordinator.

- public void registerAfter(Resource resource)
This method registers a resource (not a connection) to be registered after the registered connections are registered.

- public void registerBefore(Resource resource)
This method registers a resource (not a connection) to be registered before the registered connections are registered.

3. Identifier interface 92 is a generic interface implemented by objects that need to provide a unique identifier. In the framework of the preferred embodiment, such a unique identifier is used for the SessionID (used by the connection manager to organize the connections) and is used for the CoordinationID by the coordinator.

4. LogonInfo interface 94 defines the common way in which the infrastructure accesses the security items and provides support for security requirements on the connector and infrastructure side such as: on demand fetching of values, preloading of values, connector having an environment independent way of getting logon info and type constraints for properties. These requirements are realized by two elements in the framework of the preferred embodiment: the LogonInfo interface and the LogonInofItems class.

The implementations of the LogonInfo interface represent an infrastructure specific way of accessing security data. The LogonInfoItems class and its connector specific subclasses define which specific security items a connector is looking for. The LogonInfoItems class is to be extended by the connector by the definition of subclasses

which are specific to the security of the backend system which the connector is designed to provide a connection with.

The lookup in the infrastructure is implemented using the LogonInfo object passed on construction (as is described below, a connector retrieves the LogonInfo to use from the runtime context). Even when no further items besides user and password are needed by a particular connector, the connector has to introduce a subclass, so that a LogonInfo implementation can differentiate LogonInfoItems on a connector basis.

The LogonInfo interface defines the following methods:

- public Object getItem (LogonInfoItems logonInfoItems, String itemname)
This method retrieves a named item from the LogonInfo object.

- public void setitem (LogonInfoItems logonInfoItems, String itemname, Object item)
This method sets a named item from the LogonInfo object.

5. RASService interface 96 defines the common reliability, availability and serviceability services to all connectors. It allows error and trace logging by providing the log methods and the mechanism to retrieve the log and error streams.

The above description indicates how the CCF defines the protocols for managed connections, coordinated resources, logon information and RAS services. For a particular connector, the infrastructure must implement the interfaces ConnectionManager, Coordinator, Identifier, LogonInfo and RASService. An infrastructure which has implemented these interfaces can be used by any connector which in turn implements the interfaces of the CCF Infrastructure View, namely ManagedFactory, Managed, Resource and the class LogonInfoItems. Thus an infrastructure which implements the interfaces set out above supports all CCF connectors. The CCF Infrastructure View, as implemented on the CCF connector side, will vary for different backend systems. The implementation of the interfaces ManagedFactory, Managed, Resource and the class LogonInfoItems

will vary based on the details of the backend system to be connected to. However, once written, the connector for a particular backend will be available to be used in any CCF-supporting infrastructures.

The CCF Infrastructure View design ensures that managed connections made by a client using a CCF connector are run through a connection manager in the infrastructure and that resources are coordinated by the infrastructure. The connector itself ensures that the logon information is passed to the infrastructure by way of the accessor methods which use the LogonInfo interface.

According to the preferred embodiment, the infrastructure services are made available to a connector implementation by use of the RuntimeContext. This is a class which defines all the infrastructure services as used by the connector as shown on Figure 6.

To support CCF connectors, an infrastructure must extend the RuntimeContext class. In simple cases, the CCF itself may extend the default RuntimeContext to create a subclass which may be sufficient to for a particular infrastructure. In other cases, a subclass which is specific to the infrastructure is created. A RuntimeContext is then set up at runtime. A connector implementation then uses the RuntimeContext to obtain the infrastructure services.

The RuntimeContext defines the following methods:

- public static RuntimeContext getcurrent ()
  This method returns the runtime context associated with the current thread. If there is none it creates a default runtime context, associates it with the thread, and returns it to the caller.

- public void setCurrent (RuntimeContext runtimecontext)
  This method associates a runtime context with the current thread.

- public ConnectionManager getConnectionManager ()
  This method returns the Connection Manager of this RuntimeContext.

- public Coordinator getcoordinator ()

This method returns the Coordinator of this RuntimeContext.

- public LogonInfo getLogonInfo ()

This method returns the class containing the security information of the RuntimeContext.

- public RASService getRASService ()

This method returns the RASService of this runtime context.

- public Identifier getSessionID ()

  This method returns the unique session identifier associated with this runtime context.

- Public void close ()

  This method does the final clean up at the end of a session to handle resource and managed connections which remain in use. It also removes the RuntimeContext of the current thread.

The RuntimeContext class is used to permit the infrastructure to create a runtimecontext which is associated with the thread that the application component is running in. This thread is also the thread on which an instance of a CCF connector is called on. This mechanism of associating a runtimecontext with the thread of the connector gives the connector access to the infrastructure services at runtime.

Returning to Figure 2, the above description indicates how the definitions of the protocols between the client, connector, and interface, as set out by the Java language interface construct, permits connectors to be built which can easily run in different infrastructures and which provide a standard for different clients to access backend systems in a uniform way.

Although a preferred embodiment of the present invention has been described here in detail, it will be appreciated by those skilled in the art, that variations may be made thereto, without departing from the spirit of the invention or the scope of the appended claims.

THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. An object oriented framework for communication between an application component running in an infrastructure and a backend system, comprising

a connector protocol definition for a implementation by a connector component for communication between the application component and the backend system,

the framework further comprising

a client view protocol definition for communication between the application component and the connector component, and

an infrastructure view protocol definition for communication between the infrastructure and the connector component.

2. The object oriented framework of claim 1 in which the client view protocol definition comprises

a connection specification protocol definition to define communication of backend system connection properties to the connector component,

an interaction specification protocol definition to define communication of properties associated with application component requests of backend system,

a communication specification protocol definition to define connection, disconnection and execution of communication functions for communication between the application component and the backend system.

3. The object oriented framework of claims 1 or 2 in which the infrastructure view protocol definition comprises

a set of managed connection protocol definitions to define the creation and management of connections between the application component and the backend system, wherein the set of managed connection protocol definitions comprises a first subset of managed connection protocol definitions for implementation in the connector component and a second subset of managed connection protocol definitions for implementation in the infrastructure,

a set of resource coordination protocol definitions to define coordination of resources relating to connections between the application component and the backend system, wherein the set of resource coordination protocol definitions comprises a first subset of resource coordination protocol definitions for implementation in the connector component and a second subset of resource coordination protocol definitions for implementation in the infrastructure,

an identifier protocol definition for implementation by the infrastructure to provide connector component access to a unique identifier,

a logon information protocol definition for implementation by the infrastructure to provide connector component access to security items in the infrastructure,

a component definition for logon information for definition of security items required for connector component communication between the application component and the backend system,

a RAS protocol definition for implementation by the infrastructure to provide connector access to error and trace logging.

4. The object oriented framework of claim 3 wherein the application component runs on a thread in an infrastructure providing infrastructure services, the object oriented framework further comprising

means for the infrastructure to provide infrastructure services to the connector component by

associating a runtime context of the infrastructure services with the thread of the application component

5. The object oriented framework of claim 3 in which

the first subset of managed connection protocol definitions comprises

a protocol definition for the creation of a connection, and a protocol definition for providing a connection manager component with control over a managed connection,

and in which the second subset of managed connection protocol definitions comprises

a protocol definition for reserving and releasing managed connections.

6. The object oriented framework of claim 3 in which

the first subset of resource coordination protocol definitions comprises

a protocol definition for creating a view of a connection resource for use by a resource coordinator,

and in which the second subset of resource coordination protocol definitions comprises

a protocol definition for registration of resources by a resource coordinator.

7. The object oriented framework of claims 1, 2 or 3 in which the framework is implemented in the Java programming languages and in which the protocol definitions are interfaces in the Java programming language.

8. The object oriented framework of claim 7 in which the application component runs on a thread in a Java infrastructure, the object oriented framework further comprising means for the infrastructure to provide infrastructure services to the connector component by associating a Java runtime context with the thread of the application component

9. A computer system comprising a set of tools for use to create a communication between a first computer system component and a second computer system component, the tools constraining the communication to occur using a predefined protocol, the set of tools comprising

    a connector tool for a implementation by a connector for communication between the first computer system component and the second computer system component, the connector tool comprising

        a client view tool for defining communication between the application component and the connector component, and

        an infrastructure view tool for defining communication between the infrastructure and the connector component.

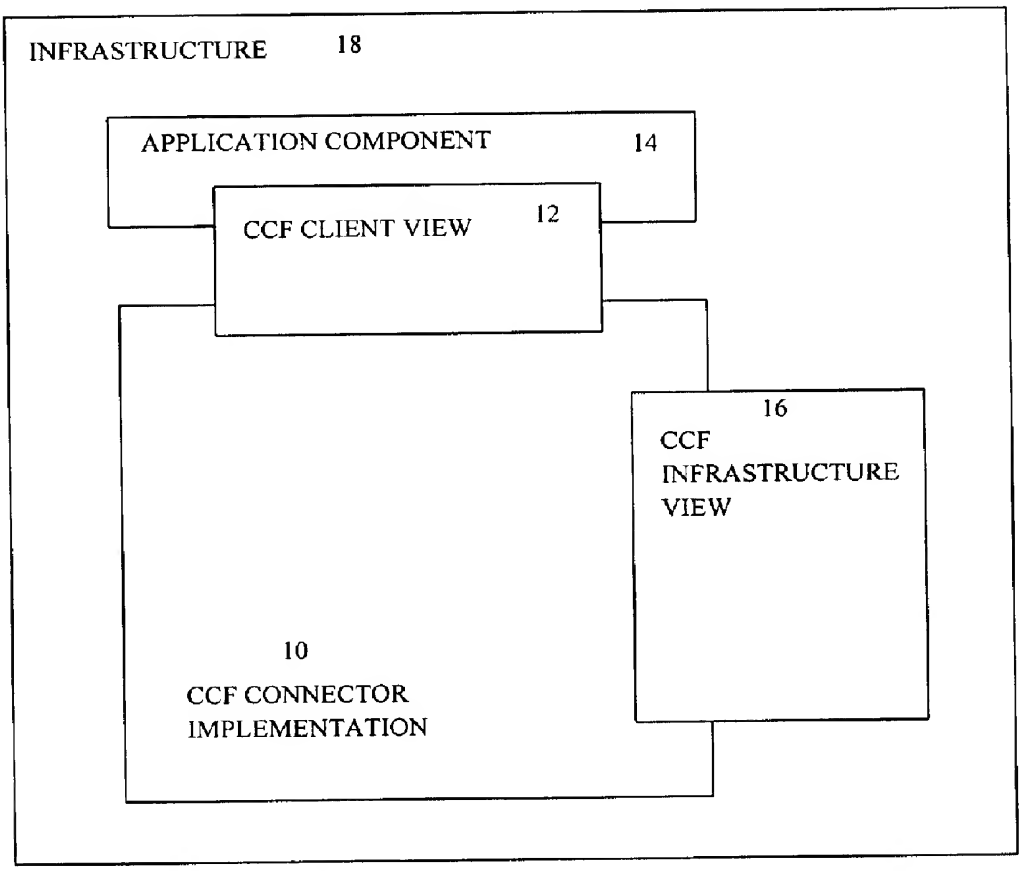

INFRASTRUCTURE 18
APPLICATION COMPONENT 14
CCF CLIENT VIEW 12
16
CCF
INFRASTRUCTURE
VIEW
10
CCF CONNECTOR
IMPLEMENTATION

FIGURE 1

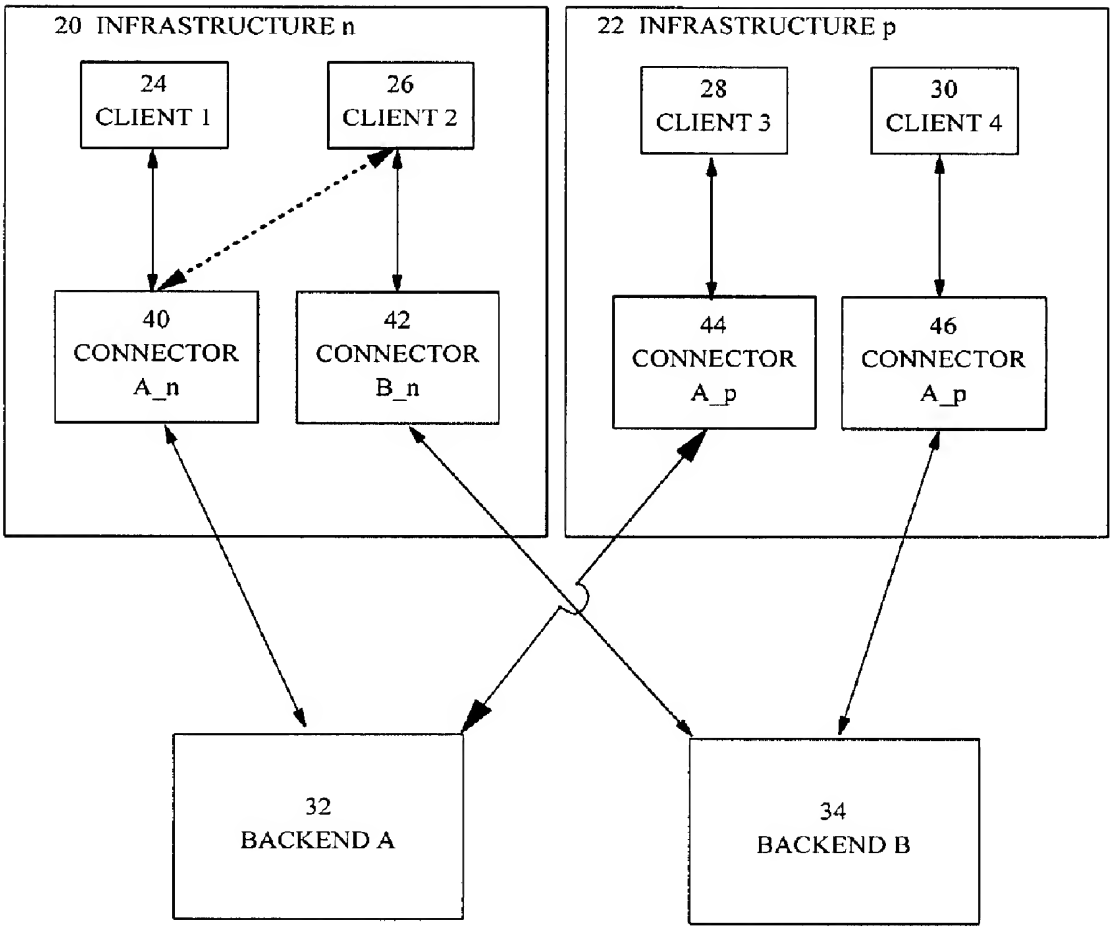

20 INFRASTRUCTURE n
24
CLIENT 1
26
CLIENT 2
40
CONNECTOR
A_n
42
CONNECTOR
B_n
22 INFRASTRUCTURE p
28
CLIENT 3
30
CLIENT 4
44
CONNECTOR
A_p
46
CONNECTOR
A_p
32
BACKEND A
34
BACKEND B

FIGURE 2

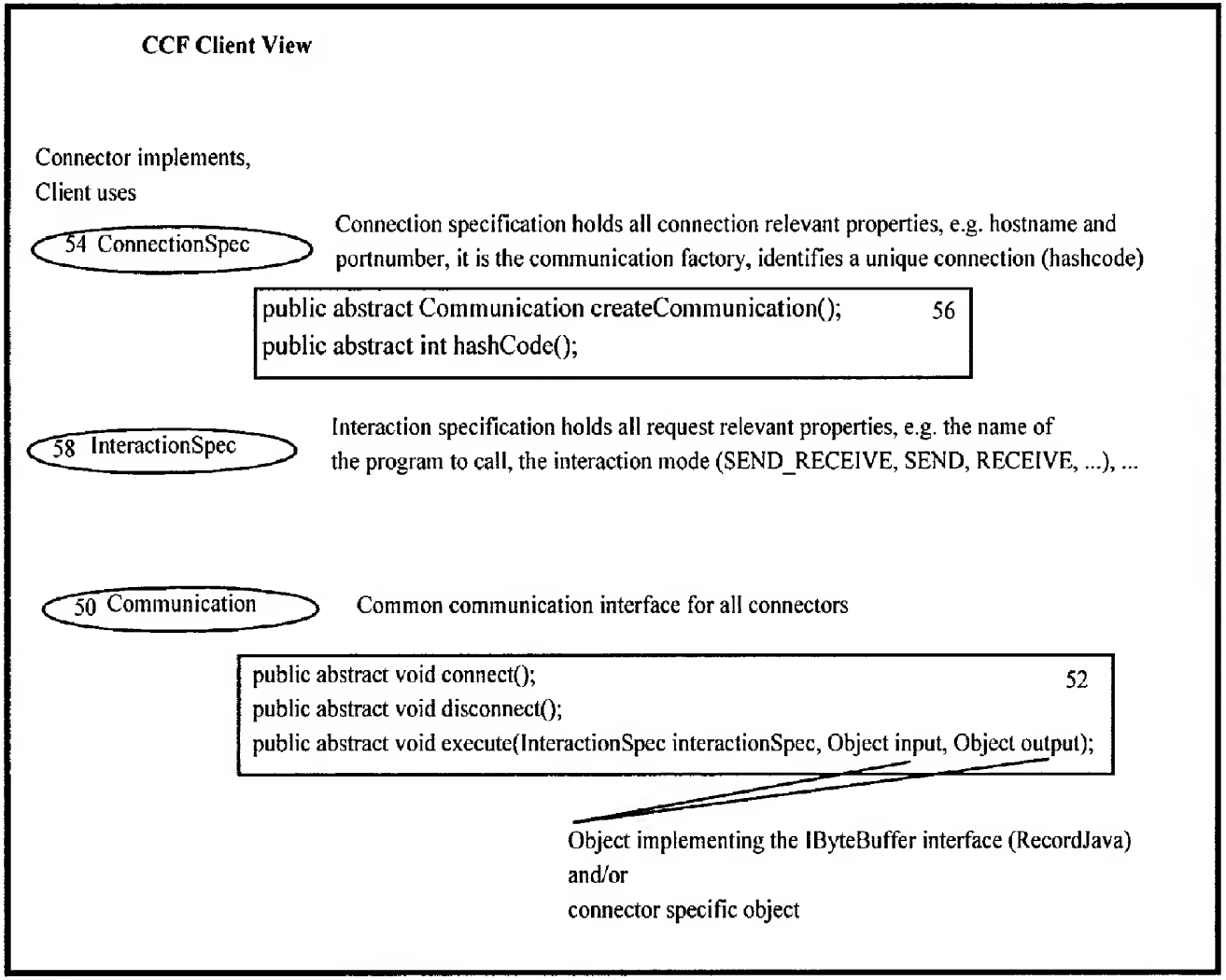


FIGURE 3

CA 02248634 1998-09-24

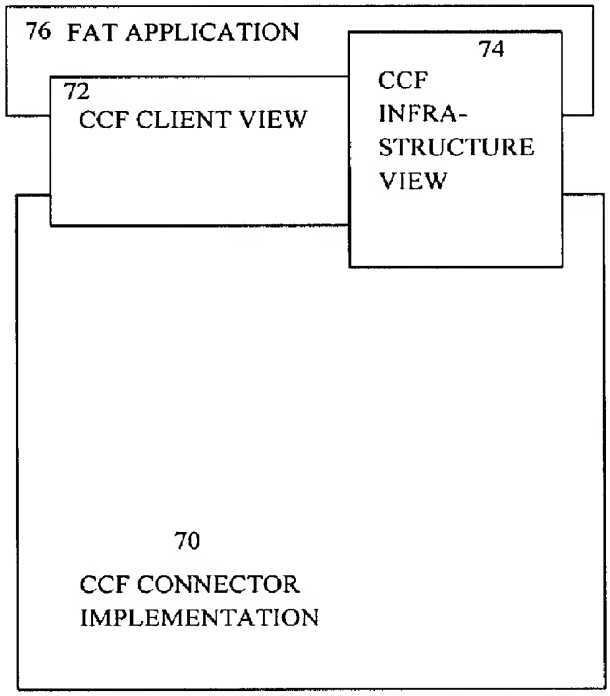

76 FAT APPLICATION
74
CCF
INFRA-
STRUCTURE
VIEW
72
CCF CLIENT VIEW
70
CCF CONNECTOR
IMPLEMENTATION

FIGURE 4

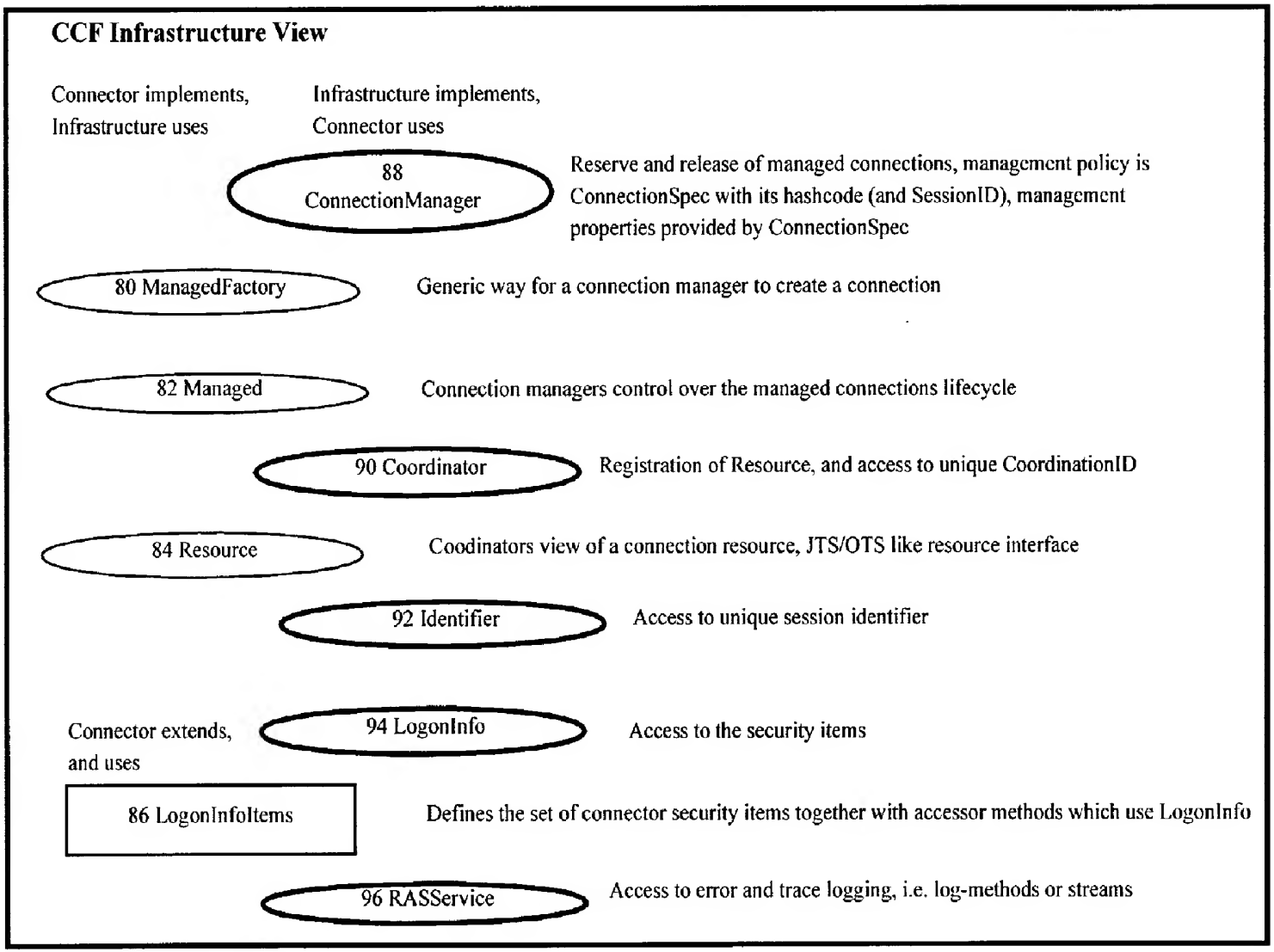


FIGURE 5

CA 02248634 1998-09-24

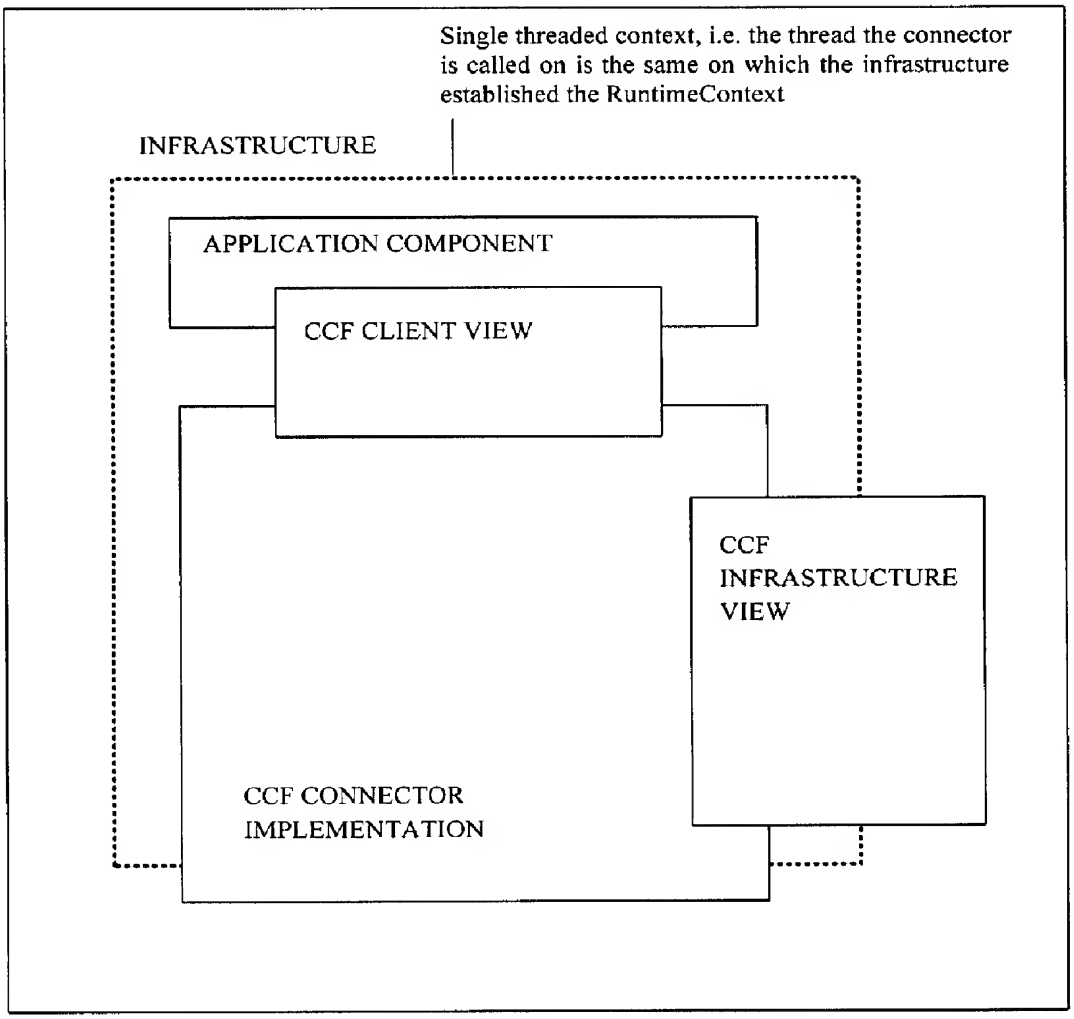

Single threaded context, i.e. the thread the connector is called on is the same on which the infrastructure established the RuntimeContext
INFRASTRUCTURE
APPLICATION COMPONENT
CCF CLIENT VIEW
CCF INFRASTRUCTURE VIEW
CCF CONNECTOR IMPLEMENTATION

FIGURE 6